How to Program like a Five-Year-Old (in Haskell)

Logic Programming and
Automatic Assembly of Effectful Computation

Jason Adaska
jwadaska@hollandhart.com

https://github.com/jadaska/assemble-lc17

How do Five-Year-Olds Program?

Puzzle-Oriented Programming (POP?)

Core Ingredients

1. *Puzzle Piece := a discrete unit of computation*
    - Programs are created by connecting pieces together
    - Pieces have edges and only “fit” together in certain ways

2. *A Five-Year-Old := Mechanism to explore possible combinations*
    - Not all combinations are valid
    - Not all combinations do the right thing (inputs + outputs)

Abstractions to make this work (Pieces)

Puzzle Pieces are Arrows

```
class Category a => Arrow a where
-- from Category | (>>>) :: a b c -> a c d -> a b d
arr :: (b -> c) -> a b c
first :: a b c -> a (b,d) (c,d)
(***) :: a b c -> a b' c' -> a (b,b') (c,c')
(&&&) :: a b c -> a b c' -> a b (c,c')
```

- Higher-kinded typeclass
  - Types on input and output
  - Constrains what can be sequenced
- E.g., Kleisli Arrow ≈ `(Monad m) => a -> m b`

Abstractions to make this work (Pieces)

```haskell
-- A wrapped arrow with type inputs and outputs
data Assembly a where
  Assembly :: (Arrow a, Typeable a, Typeable b, Typeable c)
    => a b c -> TypeRep -> TypeRep -> Assembly a

-- | A labeled arrow
data Piece a = Piece Text (Assembly a)
```

Abstractions to make this work (5 Year Old)

Five-Year-Olds are (paradoxically) the Logic Monad

- Basic idea
  - Start with puzzle pieces, input type, output type, auxiliary rules
  - Search the space until you find a valid combination or fail
  - Take first or all solutions depending on application

- Any backtracking monad (i.e., MonadPlus) works

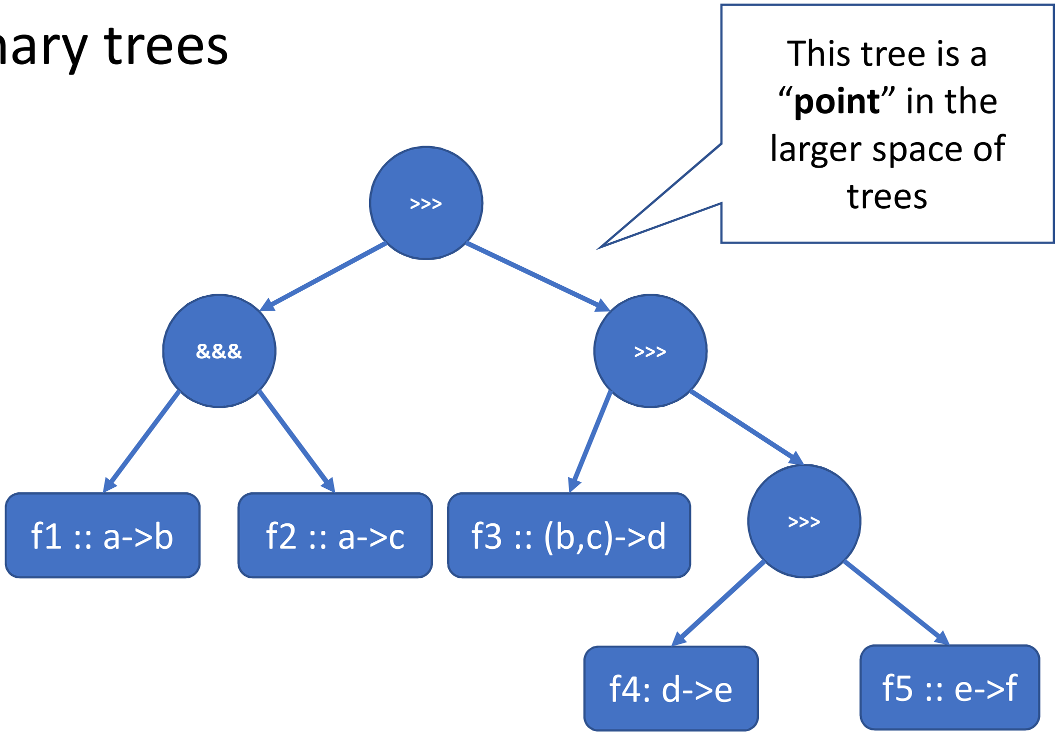
JA1

Assembly Algorithm (Hand-Waving Version)

- Arrows can be represented by a binary tree structure
- Assembly = search over valid binary trees

(f1 &&& f2) >>> f3 >>> f4 >>> f5

- f1 :: a -> b
- f2 :: a -> c
- f3 :: (b,c) -> d
- f4 :: d -> e
- f5 :: e -> f



This tree is a “**point**” in the larger space of trees

Slide 7

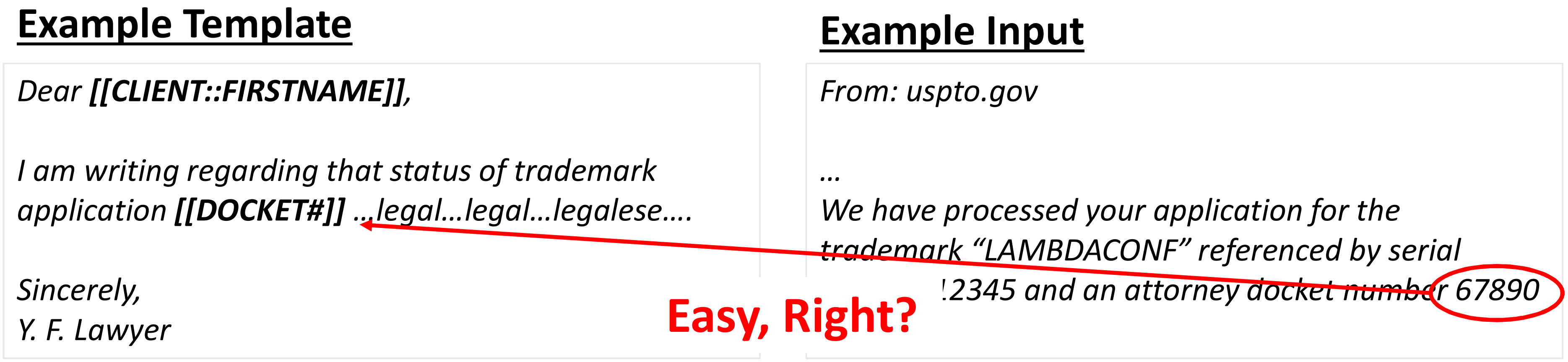
JA1 Jason Adaska, 5/27/2017

```
-- | Create an arrow with a given input type signature
fiveYearOld :: forall a . TypeRep -> StateT [Piece a] Logic (Assembly a)
fiveYearOld tr = do
    a   <- singlePiece `mplus` multipiece
    more a `mplus` return a

  where
    singlePiece :: StateT [Piece a] Logic (Assembly a)
    singlePiece = do
      pieces <- get
      piece@(Piece _ p) <- lift $ listToLogic pieces
      guard $ startsWithType p tr
      modify (removePiece piece)
      return p
      ...

-- | Use the given pieces to build an arrow with the given inputs/outpus
runFiveYearOld :: [Piece a] -> TypeRep -> TypeRep -> [Assembly a]
runFiveYearOld pieces tr1 tr2 =
  fmap fst $ observeAll
           $ runStateT m pieces
  where
    m = do
      x <- fiveYearOld tr1
      guard $ endsWithType x tr2
      return x
```

How it’s getting used : Declarative Templates



Example Template

Dear ***[[CLIENT::FIRSTNAME]]****,*

I am writing regarding that status of trademark application ***[[DOCKET#]]*** *…legal…legal…legalese….*

Sincerely,
Y. F. Lawyer

Example Input

From: uspto.gov

…

We have processed your application for the trademark “LAMBDACONF” referenced by serial 12345 and an attorney docket number 67890

Easy, Right?

But what if

- Information is incomplete
- Information is incorrect
- Parsing/NLP algorithms fail
- We want to use the same template with different input

How it’s getting used : Declarative Templates (2/2)

Six Easy Puzzle Pieces

1. Input -> DocketNum
2. Input -> SerialNum
3. Input -> ClientRefNum
4. Input -> Client
5. (Client, ClientRefNum) -> DocketNum
6. SerialNum -> DocketNum

*Function notation indicates an input/output of arrow

You Write This

Three Ways to Get DocketNum

1. Input -> DocketNum
2. (Input -> SerialNum) >>> (SerialNum -> DocketNum)
3. (Input -> Client) &&& (Input -> ClientRefNum)
   >>> ((Client, ClientRefNum) -> DocketNum)

5-Year Old (aka auto-assembly) gives you this for free

Thank you!

jwadaska@hollandhart.com

https://github.com/jadaska/assemble-lc17